Engaging End-Users in the Collaborative Development of Domain-Specific Modelling Languages

Javier Luis Cánovas Izquierdo[1], Jordi Cabot[1], Jesús J. López-Fernández[2], Jesús Sánchez Cuadrado[2], Esther Guerra[2], and Juan de Lara[2]

[1] AtlanMod, École des Mines de Nantes – INRIA – LINA, Nantes, France
{javier.canovas,jordi.cabot}@inria.fr
[2] Universidad Autónoma de Madrid, Madrid, Spain
{jesusj.lopez,jesus.sanchez.cuadrado,esther.guerra,juan.delara}@uam.es

Abstract. Domain-Specific Modelling Languages (DSMLs) are high-level languages specially designed to perform tasks in a particular domain. When developing DSMLs, the participation of end-users is normally limited to providing domain knowledge and testing the resulting language prototypes. Language developers, which are perhaps not domain experts, are therefore in control of the language development and evolution. This may cause misinterpretations which hamper the development process and the quality of the DSML. Thus, it would be beneficial to promote a more active participation of end-users in the development process of DSMLs. While current DSML workbenches are mono-user and designed for technical experts, we present a process and tool support for the example-driven, collaborative construction of DSMLs in order to engage end-users in the creation of their own languages.

Keywords: Model-Driven Engineering, Language Engineering, Domain-Specific Languages, Cooperative Engineering

1 Introduction

Model-Driven Engineering (MDE) emphasizes the use of models to raise the level of abstraction and automation in the development of software. This is achieved by defining Domain-Specific Modelling Languages (DSMLs) [1, 2] specially designed to perform tasks in certain domains, like web engineering, mobile app development, or gaming. By using DSMLs, end-users can solve problems in their domain more easily, thus becoming an important asset to improve productivity.

Interestingly enough, end-users have a very limited participation in the development of their own DSMLs. They are normally only involved in providing domain knowledge or testing the resulting language [2, 3]. This means that the MDE technical experts and not end-users (i.e., the real domain experts) are the ones in control of the DSML construction and evolution. This is a problem because errors in understanding the domain may hamper the development process

and the quality of the resulting DSML. Thus, it would be beneficial to promote a more active participation of end-users in the DSML development process.

To make effective this participation, some technical barriers need to be overcome and means to foster the collaboration in the community of end-users of the DSML are needed. First, end-users should be liberated from doing development tasks requiring too technical, specialized MDE abilities (e.g., the identification of abstract concepts or their implementation in platform-specific meta-models). To this aim, we propose supporting language development by means of examples [4]. Thus, users are able to draft language examples from which a language definition (i.e., language domain and syntax) can be automatically derived. Second, given that a language targets a community of end-users, it is crucial to drive the participation of its members in a collaborative fashion where each member cooperates with their peers to make the language development process progress. The discussions arisen as a result of this participation drive the development process and become a valuable documentation of the design decisions [5].

For this purpose, this paper provides an approach for the example-driven, collaborative construction of DSMLs, which combines the works described in [4, 5]. We propose an iterative process, which starts with a set of examples that are refined in each iteration. Refinement consists of providing language modifications by means of new examples, which are discussed by the end-users to reach agreement to drive the language development process. Furthermore, to help making design decisions, the approach also incorporates a recommender system which identifies and proposes changes in the language according to meta-model quality patterns [6–8].

The rest of the paper is organized as follows. Section 2 describes how DSMLs are built nowadays, using a running example. Section 3 overviews our approach and Section 4 its main technical aspects. Section 5 compares with related work and Section 6 concludes the paper.

2 Engineering Domain-Specific Languages

DSMLs are languages tailored to a specific task or domain, capturing its main primitives and abstractions [1, 2]. Examples of DSMLs include dedicated languages for web engineering, requirements specification, business modelling, or data querying. DSMLs are not only specific to computer science, but are useful in many diverse areas and disciplines, like biology, physics, management or education, where the domain experts are not necessarily computer scientists, and may not have knowledge of MDE platforms and tools.

A DSML is defined by its abstract syntax, concrete syntax and semantics. The abstract syntax describes the concepts of the domain, their features and relations. In MDE, the abstract syntax is built through a meta-model, normally a class diagram with additional constraints. The concrete syntax describes the representation of models, either graphically (e.g., an electrical circuit), textually (e.g., an SQL query), or a combination of both. The semantics describes the meaning of models by providing, e.g., a description of their execution, or a

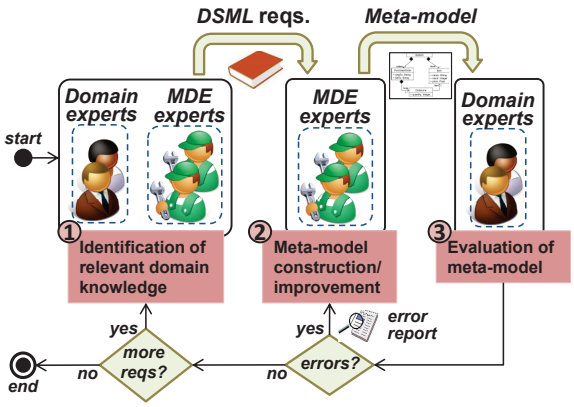

Fig. 1. Traditional process for meta-model construction.

mapping into a semantic domain. While a DSML covers these three aspects, in this paper we concentrate on the process of constructing the abstract syntax.

There are many workbenches for developing DSML editors [1, 2, 9], but they suffer from some drawbacks. First, they are directed to computer scientists with a background in MDE, as the first activity is creating a meta-model. While building meta-models is natural to software engineers, it can be challenging for end-users, who may prefer drafting example models first, share and discuss them with their peers, and only then abstract them in a meta-model. Asking end-users to build a meta-model *before* drafting examples is often too demanding. Moreover, these frameworks do not foster the active participation of the domain experts in the DSML design process. Their role is limited to providing background knowledge of the domain, and evaluating the DSML proposals created by the MDE experts. This may lead to misunderstandings of the domain concepts, omissions or non-optimal solutions.

A scheme of this "traditional" process is shown in Figure 1: first, there is a requirements gathering meeting *(1)*; then, a meta-model is built *(2)*; and next, this meta-model is reviewed by the end-users *(3)*. If defects are found, feedback is provided and the meta-model is reworked. This process is iterated until the meta-model gathers all domain concepts and then the DSML tooling can be developed. Note that, sometimes, defects can only be found once the tooling is available, when end-users detect missing elements, thus requiring rebooting the process.

As an example, consider the construction of a DSML to describe the passenger entry process of an airport, with the purpose of performing a queue-based simulation to optimize this process. The stakeholders include therefore terminal operator supervisors, airport management staff, and terminal managers of specific airlines. A possible meta-model is shown in Figure 2(a), where the check-in and airplane queues are identified.

The traditional process has some drawbacks at every stage. In step *(1)*, domain knowledge is documented in natural language. However, having this knowl-

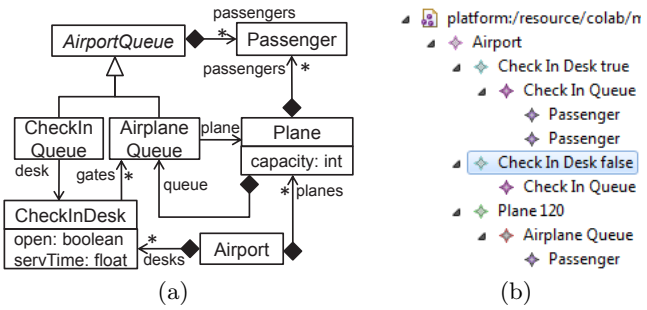

Fig. 2. (a) Sample meta-model. (b) Sample model.

edge in the form of computer-processable models would be more effective. Unfortunately, this is not possible as, at this point, no meta-model exists yet. Second, evaluating the meta-model in step *(3)* is difficult, as the end-users may need to build testing models using "raw" abstract syntax, with no intuitive support for the concrete syntax. For example, in the case of Eclipse EMF [10], end-users need to build models using a tree-based editor (see Figure 2(b)). Moreover, domain experts are demanded to understand a meta-model that includes conceptual modelling elements like inheritance, composition and textual constraints, which might be difficult to grasp to non-experts. In addition, some meta-model elements, like concept Airport, are only needed due to the implementation platform of the meta-model. For example, EMF requires a root concept in order to generate a tree-based editor. Finally, this process may lead to heavy iterations that could be optimized with a more active involvement of end-users in the meta-model design, at stage *(2)*, which in turn could help to get a meta-model fitting the end-users' needs and ready to be used to develop the DSML tooling. Hence, in the next section, we present a collaborative process aiming at alleviating these problems.

3 A Collaborative Process Driven by Examples

The drawbacks identified in the previous section prevent end-users from participating effectively in the creation of DSMLs. To promote their engagement, we propose a collaborative development process driven by examples. The use of example models liberates end-users from doing too demanding technical tasks for their expertise (e.g., defining abstract concepts), thus enabling their active participation in the process. Moreover, the development process evolves in a collaborative fashion where any end-user cooperates and discusses about the changes to be made in the language. End-users involved in the creation of the DSML become *the community*, and as a result of their collaboration, the community as a whole decides the changes that will be eventually added to the DSML.

We propose a process to build the abstract syntax of DSMLs composed of five phases: (1) process bootstrapping, (2) meta-model induction, (3) evaluation and

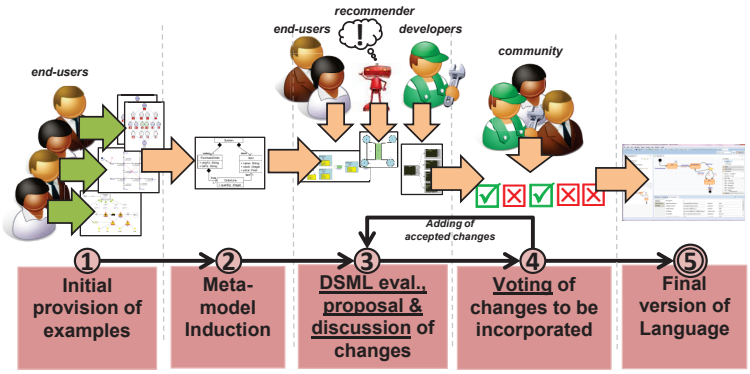

Fig. 3. Process for building DSMLs collaboratively driven by examples.

discussion, (4) voting phase and (5) language development. Next, we describe in detail each phase, which are illustrated in Fig. 3.

To bootstrap the process, end-users initially provide a set of examples (full models or fragments) which illustrate the DSML to develop (see step *(1)*). These examples are only sketches (i.e., they do not conform to any meta-model) where icons are named and arranged, but give the end-users the power of tailoring the DSML to their needs without performing too demanding tasks. A reasoning procedure on these examples generates automatically the abstract syntax of the language (step *(2)*), which is defined by means of a meta-modelling language. In our case, we use the Ecore meta-modelling language, thereby abstract syntax models conform to the Ecore meta-model, where concepts are represented as instances of `EClass` elements and their attributes and references are represented as instances of `EAttribute` and `EReference` elements, respectively. Once a first version of the language is obtained from the examples, an iterative process starts to collaboratively develop and refine the language.

In the next phase (step *(3)*), the generated meta-model is evaluated through examples, which may trigger the proposal and discussion of changes. The DSML can evolve due to three main inputs: (1) community members, (2) virtual assistant and (3) technical experts. Community members can propose ideas or changes to the DSML, e.g., after checking the examples they can ask for modifications on the language. A change proposal includes a description of the problem and, optionally, a set of new model examples illustrating the issue. Internally, the changes to be performed in the meta-model to accommodate the new model examples are automatically derived, thus liberating end-users from the task of devising how the language definition should be modified to include their proposals. A virtual assistant can also propose some improvements based on a set of predefined design patterns, thereby assisting end-users in increasing the quality of the DSML. Finally, technical experts can also collaborate to support the language definition process. All change proposals can then be discussed and eventually accepted/rejected.

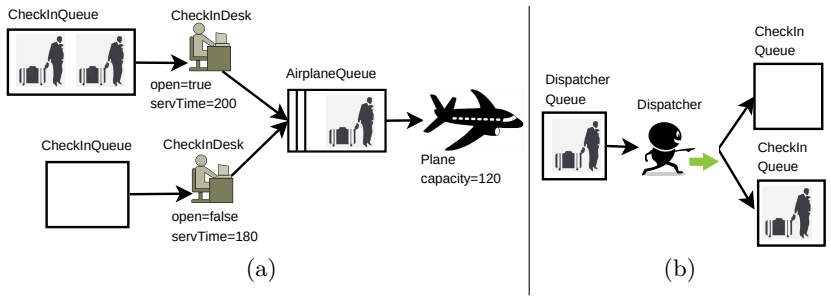

Fig. 4. (a) Initial example. (b) Corrective example.

The decision to accept or reject a change proposal depends on whether the community reaches an agreement. For this purpose, community members can vote the proposal (step *(4)*). A decision engine analyses the votes committed to calculate (according to agreed collaboration rules such as majority, unanimity, etc.) which proposals are accepted/rejected. The accepted proposals are then incorporated into the language and a new iteration is performed. The process keeps iterating until no more changes are proposed. At the end of the process, the language can be implemented with the sureness that the meta-model fulfils the end-users' needs (step *(5)*).

To illustrate the process, Figure 4 shows the outcomes of a possible collaboration scenario for the running example introduced in Section 2. The process starts with a set of model fragments from which a first version of the meta-model is automatically obtained. One of the provided examples is shown in Figure 4(a), which is actually the same as the one in Figure 2(b) but using a much more intuitive, user-friendly graphical syntax. The obtained meta-model matches with the one presented in Figure 2(a). The meta-model and the examples are then shared in the community to be validated. Validation can be done by looking at the existing examples but also giving new ones and then an automated procedure checks whether those fragments are accepted by the current meta-model.

At some point, an end-user detects a problem because the concept of Dispatcher (a welcome agent that redirects passengers to appropriate checkin queues) is not included so he submits a corrective model fragment including this concept (see Figure 4(b)). Examples do not need to be full-fledge models, but may be partial models focussing on some interesting aspect. The new example becomes a new change proposal to which an automatic process attaches the modifications that should be performed in the first version of the meta-model. This change proposal must be validated by the community, who vote for/against it. Let us consider that eventually it is accepted, consequently, the change proposal is realised, resulting in the meta-model shown in Figure 5(a).

Concurrently, technical experts and the virtual assistant can also propose modifications. For instance, a technical expert can detect the need of including an attribute in the concept CheckInDesk to specify the initialization time. This change is also proposed to the community, who eventually agrees on its incorporation into the language (see Figure 5(b)). On the other hand, the vir-

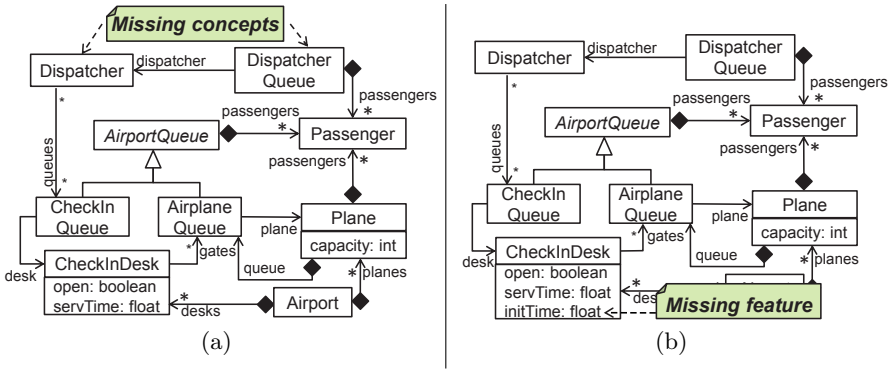

Fig. 5. (a) Resulting meta-model after discussing the corrective example. (b) Resulting meta-model after including changes from technical experts.

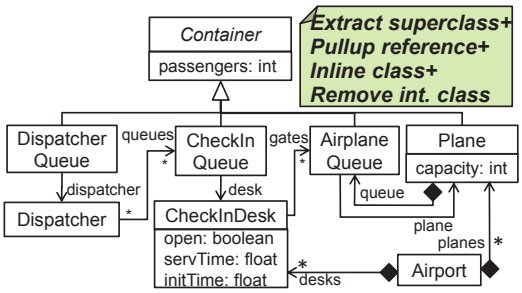

Fig. 6. Resulting meta-model after including the change proposals of the virtual assistant.

tual assistant analyses each version of the meta-model to recommend possible improvements and refactoring opportunities. For instance, since Dispatcher, AirportQueue and Plane have a reference to Passenger, the assistant recommends creating a new concept as superclass of the first three elements thus factorizing the reference. This recommendation is included in the process as a new change proposal so it is also submitted for debate and eventually for acceptance/rejection. Actually, the recommender also proposed to inline class Passenger (as it does not have attributes) creating the integer attribute passengers, and to eliminate the abstract class AirportQueue (see Figure 6).

4 Technical Solution

The proposed process has been implemented on top of Collaboro [5], an Eclipse-based tool to develop Ecore-based DSMLs collaboratively. Collaboro allows modelling the collaborations among community members when developing a DSML: proposals describe language changes, solutions specify how changes should be implemented in the language (e.g., adding concepts, removing attributes, etc.), and comments can be added to both of them. The tool also allows community

members to vote change proposals and includes a decision engine which analyses these votes to calculate which collaborations are eventually accepted/rejected. However, Collaboro only allows modelling collaborations at the meta-model level (i.e., in terms of abstract elements). Our proposal extends the tool to support the collaborative definition of DSMLs by means of examples.

The example-based induction of meta-models is realized using the *bottomUp* tool we presented in [4], which was integrated in Collaboro. As Collaboro needs a model of the changes to be incorporated to the meta-model, we extended the *bottomUp* tool with: (i) a mechanism to record and serialize the changes produced by the induction algorithm, (ii) a virtual assistant and a mechanism to record and serialize the changes produced by its recommendations, and (iii) an importer of Ecore meta-models. The latter is needed in order to process the current version of the meta-model, provided by Collaboro. Collaboro was therefore extended to integrate both the *bottomUp* tool and the virtual assistant as part of the DSML definition process.

Figure 7 shows a snapshot of our tool when the end-user proposes the corrective fragment in the running example. The snapshot includes the meta-model automatically generated from the initial example (left) and the change proposal created by the end-user. The change proposal includes a reference to the file with the fragment (as a child element of the proposal), as well as the set of changes to be done in the meta-model (right bottom part), which are automatically derived from the fragment. The snapshot also shows the contextual menu which allows voting for/against a collaboration as well as commenting them (left top part).

Our implementation supports defining examples as DIA[1] diagrams, as it contains a rich palette of over 1000 icons. End-users can therefore create new proposals and attach the corresponding diagram. For each example-based proposal, Collaboro invokes *bottomUp* to automatically derive the changes to perform in the meta-model of the language, thus creating a solution in the proposal describing these changes. These proposals are then shared with the community to be voted and eventually accepted/rejected. If accepted, they are incorporated to the meta-model by Collaboro.

Our approach also includes a virtual assistant, which analyses the abstract syntax meta-model under development and recommends possible improvements. The implementation of the assistant is integrated in *bottomUp* and incorporates some heuristics to detect errors and modifications. These include refactorings [8] like extracting a common super-class when a set of classes share features, different ways to in-lining or merge classes, and eliminating intermediate abstract classes.

5 Related work

End-user collaboration is a key feature in software development methods such as agile-based ones as well as in user-centered design [11]. The advantages of

[1] http://dia-installer.de

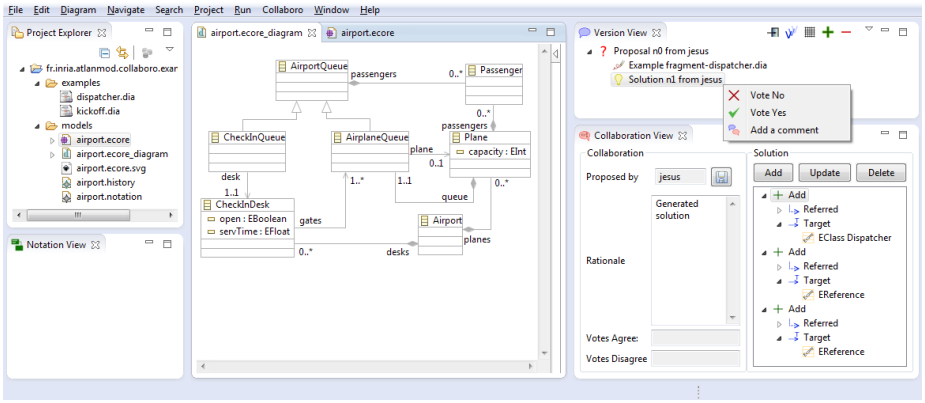

Fig. 7. Induced meta-model (left). Providing example-based proposals in Collaboro (right).

collaborating in the development of software have been studied in works such as [12, 13] and they are also illustrated in fields such as requirements elicitation [14] or global software development [15] but neither of them focus on creating DSMLs.

In [16], a collaborative modelling environment is presented, based on EMF Eclipse framework. While it supports the collaborative use of modelling languages, it does not support their collaborative construction. On the other hand, the COMA tool [17] allows collaborating in the definition of UML diagrams, however, it does not provide support for creating DSMLs and does not present the collaboration as a process of discussion, argumentation and voting.

While several works emphasize the benefits of using examples when developing modelling abstractions [18], our proposal is unique in combining an example-based and collaborative approach to define DSMLs.

6 Conclusions

In this paper we have presented a process and tool support[2] for the collaborative development of DSMLs, where end-users are engaged and play an active role in the development of their own language. This is possible by the use of example models and fragments as a mechanism to drive the process, on the explicit support for discussion and collaboration, and on automated technical advice by a virtual recommender system.

In the future, we plan to apply our approach in the context of projects with our industrial partners. This will enable an empirical study of our solution, with the goal of checking whether end-users actually prefer it instead of the existing traditional solutions for defining DSMLs, and assess the quality of the produced DSML. We also plan to provide support for the automatic detection and solution

[2] https://code.google.com/a/eclipselabs.org/p/collaboro

of conflicting change applications, and for the collaborative construction of the concrete syntax and semantics of DSMLs, resulting in the generation of a full-fledged modelling environment.

Acknowledgements. This work was funded by the Spanish Ministry of Economy and Competitivity (project "Go Lite" TIN2011-24139), the R&D programme of the Madrid Region (project "e-Madrid" S2009/TIC-1650) and the European Commission under the ICT Policy Support Programme, grant no. 317859.

References

1. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE CS (2008)
2. Voelter, M.: DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. CreateSpace (2013)
3. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37** (2005) 316–344
4. Cuadrado, J.S., de Lara, J., Guerra, E.: Bottom-up meta-modelling: An interactive approach. In: MODELS conf. Volume 7590 of LNCS., Springer (2012) 3–19
5. Cánovas Izquierdo, J.L., Cabot, J.: Enabling the collaborative definition of DSMLs. In: CAiSE conf., To appear (2013)
6. Aguilera, D., Gómez, C., Olivé, A.: A method for the definition and treatment of conceptual schema quality issues. In: ER. Volume 7532 of LNCS., Springer (2012) 501–514
7. Cho, H., Gray, J.: Design patterns for metamodels. In: DSM'11. (2011)
8. Fowler: Refactoring: Improving the Design of Existing Code. Addison-Wesley (1999)
9. Xtext. http://www.eclipse.org/Xtext/
10. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, 2^{nd} Edition. Addison-Wesley Professional, Upper Saddle River, NJ (2008)
11. Norman, D.A., Draper, S.W.: User Centered System Design: New Perspectives on Human-computer Interaction. Erlbaum, Hillsdale (1986)
12. Hildenbrand, T., Rothlauf, F., Geisser, M., Heinzl, A., Kude, T.: Approaches to collaborative software development. In: FOSE conf., IEEE (2008) 523–528
13. Whitehead, J.: Collaboration in software engineering: A roadmap. In: FOSE conf., IEEE (2007) 214–225
14. Mylopoulos, J., Chung, L., Yu, E.: From Object-Oriented to Goal-Oriented Requirements Analysis. Commun. ACM **42**(1) (1999) 31–37
15. Lanubile, F., Ebert, C., Prikladnicki, R., Vizcaino, A.: Collaboration tools for global software engineering. IEEE Softw. **27**(2) (2010) 52–55
16. Gallardo, J., Bravo, C., Redondo, M.A.: A model-driven development method for collaborative modeling tools. J. Network and Computer Applications **35**(3) (2012) 1086–1105
17. Rittgen, P.: COMA: A tool for collaborative modeling. In: CAiSE Forum. (2008) 61–64
18. Bak, K., Zayan, D., Czarnecki, K., Antikewicz, M., Diskin, Z., Wasowski, A., Rayside, D.: Example-Driven Modeling. Model = Abstractions + Examples. In: NIER track at ICSE'13. (2013)